AIML ZG537
Information Retrieval

Dr. Maheswari Karthikeyan

Lecture3

BITS Pilani
Pilani Campus

Lecture Outline

innovate achieve lead

Boolean Retrieval (Chapter 2)

- Term vocabulary and Postings lists
- Phrase Queries

Slides are from Introduction to Information Retrieval;Manning,Raghavan

http://nlp.stanford.edu/IR-book

Term Vocabulary & Postings List

Term Vocabulary & Postings List

innovate achieve lead

- Term vocabulary
  - Documents
  - Tokenization
  - Linguistic Pre-processing of tokens
  - Indexing
- Postings
  - Faster merges: skip lists
  - Positional postings and phrase queries

Document delineation

- **Obtaining the character sequence in a document**
- **Choosing a document unit**
  - We need to deal with format and language of each document.
  - What format is it in? pdf, word, excel, html etc.
  - What language is it in?
  - What character set is in use?

Vocabulary of Terms

innovate achieve lead

- **Tokenization**
- Stop words
- Normalization
- Stemming and Lemmatization

Tokenization

- It is the task of chopping the document into pieces, called ***tokens***, by removing unwanted characters

Example of tokenization:

Input: "***Friends, Romans, Countrymen***"

Output:

Tokens

Friends

Romans

Countrymen

- Each such token is now a candidate for an index entry, after further processing.

Tokenization

innovate achieve lead

Token: A *token* is an instance of a sequence of characters in a particular document that are grouped together as a useful semantic unit for processing

Type:. A *type* is the class of all tokens containing the same character sequence.

Term : A *term* is a (perhaps normalized) type that is included in the IR system's dictionary.

"She was young the way an actual young person is young."

11 tokens, 9 types, 5 terms

Issues in Tokenization

innovate achieve lead

Cooper's concordance of Wordsworth was published in 1911. The applications of full-text retrieval are legion: they include résumé scanning, litigation support and searching published journals on-line.

- ***Cooper's*** vs. ***Cooper*** vs. ***Coopers***.
- ***Full-text*** vs. ***full text*** vs. {***full, text***} vs. ***fulltext.***
- Accents: ***résumé*** vs. ***resume***.

Issues in Tokenization

innovate achieve lead

What are the correct tokens to use?

- **Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.**

Language specific

Distinctive signature patterns

Issues in Tokenization

innovate achieve lead

- **Hyphens**
  - Hewlett-Packard
  - Hewlett and Packard as two tokens?
  - state-of-the-art:
  - co-education
  - lowercase, lower-case, lower case
- **White Space**
  - ***San Francisco***: one token or two?

Issues in Tokenization

- Different character sequences
  - email addresses (jblack@mail.yahoo.com)
  - Web URLs (http://stuff.big.com/new/specials.html)
  - numeric IP addresses (142.32.48.231)
  - package tracking numbers (1Z9999W99845399981)

- Often have embedded spaces
- Older IR systems may not index numbers

But often very useful:

- **looking up error codes/stack traces on the web**
- **Date of an email :Will often index “meta-data” separately, Creation date, format, etc.**

Issues in Tokenization

- **Tokenization: language issues**
- French
  - ***L'ensemble*** one token or two?
  - ***L*** ? ***L'*** ? ***Le*** ?
  - Want ***l'ensemble*** to match with ***un ensemble***
- German noun compounds are not segmented
  - ***Lebensversicherungsgesellschaftsangestellter***
  - 'life insurance company employee'

Issues in Tokenization

- Chinese and Japanese have no spaces between words:

  莎拉波娃现在居住在美国东南部的佛罗里达。

  - Not always guaranteed a unique tokenization
- Further complicated in Japanese, with multiple alphabets intermingled
  - Dates/amounts in multiple formats
- Arabic (or Hebrew) is basically written right to left, but with certain items like numbers written left to right
  - Words are separated, but letter forms within a word form complex ligatures

Vocabulary of Terms

- Tokenization
- **Stop words**
- Normalization
- Stemming and Lemmatization

Stop words

- Stop words: common words, which are of little value in selecting documents, matching a user need
- Stop words have little semantic content:
  - *the, a, and, to, be*
- Stop list: the common words to be discarded while indexing
- Sort the terms by *collection frequency* and then take the most frequent.

Collection frequency : the total number of times each term appears in the document collection.

Stop words

innovate achieve lead

Issues in removing stop words

- Some special query types are disproportionately affected
  - **Phrase queries:**
    - "King of Denmark"
    - "President of the United States", President AND "United States
  - **Various song titles, etc.:**
    - "Let it be", "To be or not to be"
  - "**Relational" queries:**
    - "flights to London"

Stop words

innovate achieve lead

Issues in removing stop words

- Standard use of quite large stop lists (200–300 terms) to very small stop lists (7–12 terms)
- Good compression techniques means the space for including stop words in a system is very small
- Good query optimization techniques means to pay little at query time for including stop words.

- https://www.ranks.nl/stopwords

Vocabulary of Terms

- Tokenization
- Stop words
- **Normalization**
- Stemming and Lemmatization

Normalization

- *Token normalization* is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens
  - **We want to match *U.S.A.* and *USA***
- Result is terms: a term is a (normalized) word type, which is an entry in the IR system dictionary
- Standard way is to implicitly create *equivalence classes*, which are normally named after one member of the set

e.g.,

- deleting periods to form a term
  - ***U.S.A., USA***
- deleting hyphens to form a term
  - ***anti-discriminatory, antidiscriminatory***

- An alternative to creating equivalence classes is to maintain relations between unnormalized tokens.

Normalization

- Handling synonyms and homonyms
  - E.g., by hand-constructed equivalence classes
    - ***car*** = ***automobile*** ***color*** = ***colour***
  - rewrite to form equivalence-class terms
    - When the document contains ***automobile***, index it under ***car-automobile*** (and vice-versa)
  - Or expand a query
    - When the query contains ***automobile***, look under ***car*** as well
- Spelling mistakes
  - One approach is Soundex, which forms equivalence classes of words based on phonetic heuristics

Normalization

- There is asymmetry in expansion
- An example of where this may be useful
  - Enter: ***window*** Search: ***window, windows***
  - Enter: ***windows*** Search: ***Windows, windows, window***
  - Enter: ***Windows*** Search: ***Windows***

Normalization

innovate achieve lead

- Accents and Diacritics
- Case folding – True Casing
- Reduce all letters to lower case
- The simplest heuristic is to convert to lowercase words at the beginning of a sentence and all words occurring in a title that is all uppercase or in which most or all words are capitalized
  - exception: upper case in mid-sentence

Vocabulary of Terms

- Tokenization
- Stop words
- Normalization
- **Stemming and Lemmatization**

Stemming and Lemmatization

- To reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.

E.g.,

- *am, are, is → be*
- *car, cars, car's, cars' → car*

the boy's cars are different colors →

the boy car be different color

Stemming and Lemmatization

innovate achieve lead

- ***Stemming*** refers to a crude heuristic process that chops off the ends of words and removes the derivational affixes.
- It commonly collapses derivationally related words.
- ***Lemmatization*** refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*.
- It only collapses the different inflectional forms of a lemma.

Stemming

Reduce terms to their “roots” before indexing

“Stemming” suggests crude affix chopping

- language dependent
- e.g., ***automate(s), automatic, automation*** all reduced to ***automat***.

for example compressed and compression are both accepted as equivalent to compress.

for exampl compress and compress ar both accept as equival to compress

Porter's algorithm

The most common algorithm for stemming English

- Results suggest it's at least as good as other stemming options

Algorithm

- 5 phases of reductions
- phases applied sequentially
- each phase has various conventions to select rules
- sample convention: Of the rules in a compound command, select the one that applies to the longest suffix.

Porter's algorithm

sses → *ss*

ies → *i*

ational → *ate*

tional → *tion*

Weight of word sensitive rules

(m>1) EMENT →

- *replacement* → *replac*
- *cement* → *cement*

Other Stemmer Algorithms

- Lovins stemmer
  - http://www.comp.lancs.ac.uk/computing/research/stemming/general/lovins.htm
  - Single-pass, longest suffix removal (about 250 rules)
- Paice/Husk stemmer
- Snowball

A comparison of three stemming algorithms on a sample text

Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Lovins stemmer: such an analys can reve featur that ar not eas vis from th vari in th individu gen and can lead to a pictur of expres that is mor biolog transpar and acces to interpres

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Paice stemmer: such an analys can rev feat that are not easy vis from the vary in the individ gen and can lead to a pict of express that is mor biolog transp and access to interpret

Lemmatizer

- Tool from Natural language Processing which does full morphological analysis to accurately identify the lemma for each word. Doing full morphological analysis produces at most very modest benefits for retrieval.

Stemmer and Lemmatizer

- The above methods embody transformations that are
  - Language-specific, and often
  - Application-specific
- These are “plug-in” addenda to the indexing process
- Both open source and commercial plug-ins are available for handling these.
- https://text-processing.com/demo/stem/

What does Google do?

- Stop words
- Normalization
- Tokenization
- Lowercasing
- Stemming
- Umlauts
- Compounds
- Numbers

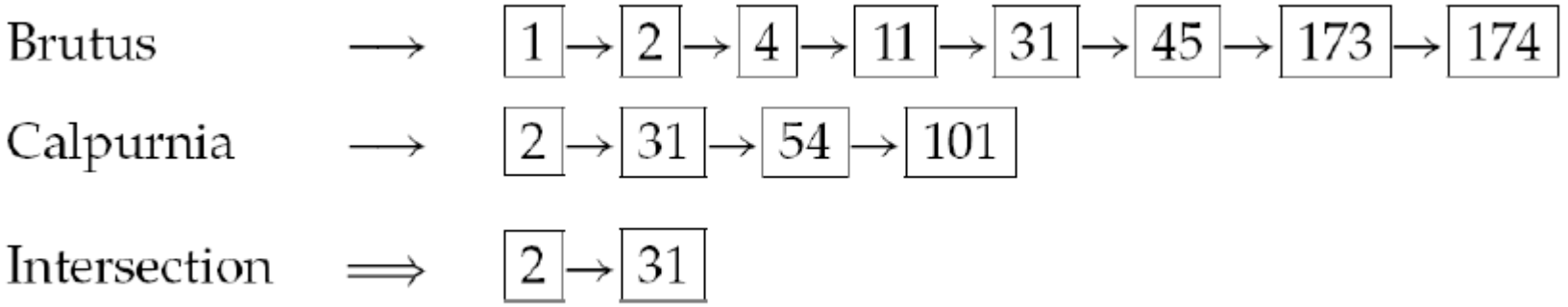
Postings List

Postings

- Faster merges: skip lists
- Positional postings and phrase queries

Faster postings list

- **Faster postings list intersection via skip pointers**
  - ways to increase the efficiency of using



- If the list lengths are *m* and *n*, the merge takes O(*m*+*n*) operations.

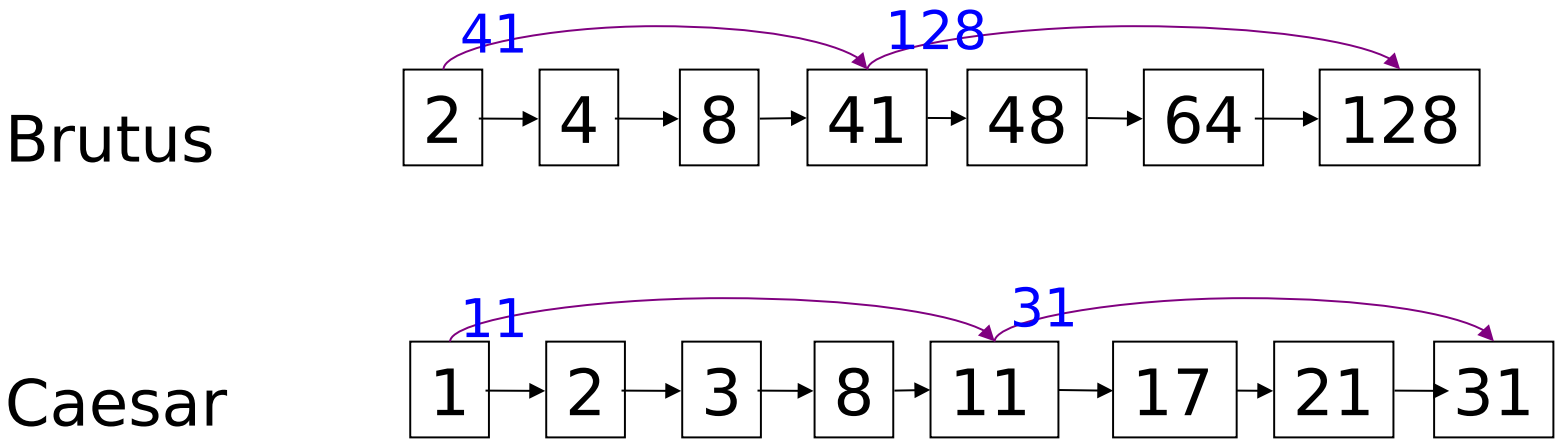
Skip Pointers

innovate achieve lead

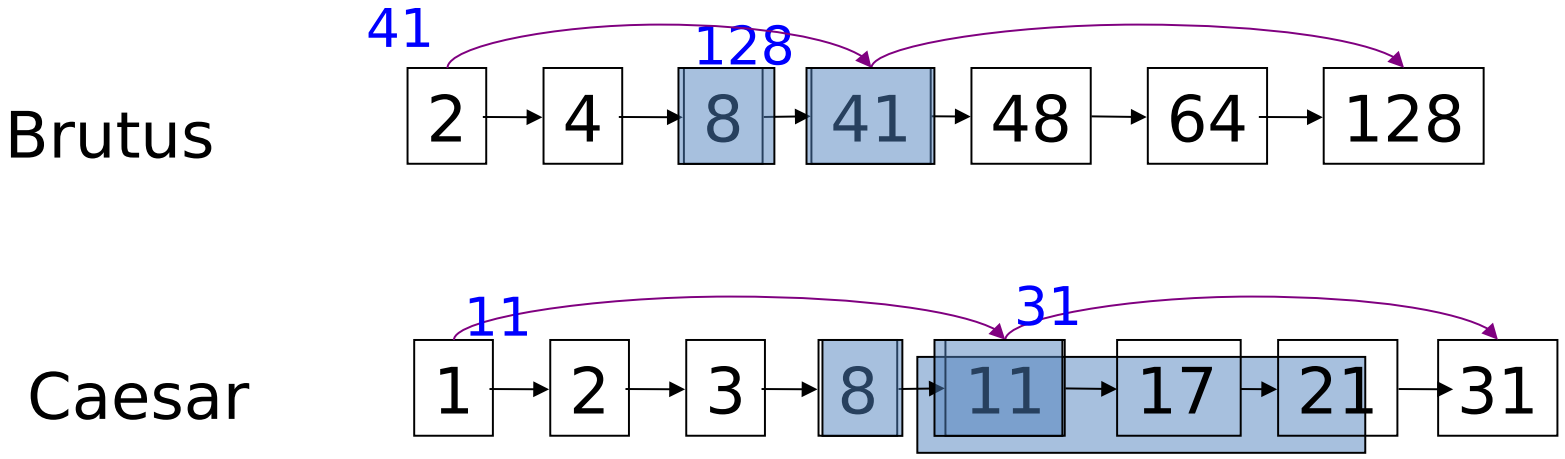
- Augmenting postings lists with skip pointers
- Skip pointers are effectively shortcuts that allow us to avoid processing parts of the postings list that will not figure in the search results.

- **Where to place skip pointers ?**
- **How to do efficient merging using skip pointers?**

Skip Pointers



Skip Pointers



- Suppose we've stepped through the lists until we process **8** on each list. We match it and advance.
- We then have **41** and **11** on the lower. **11** is smaller
- But the skip successor of **11** on the lower list is **31**, so we can skip ahead past the intervening postings.

Postings lists intersection with skip pointers.

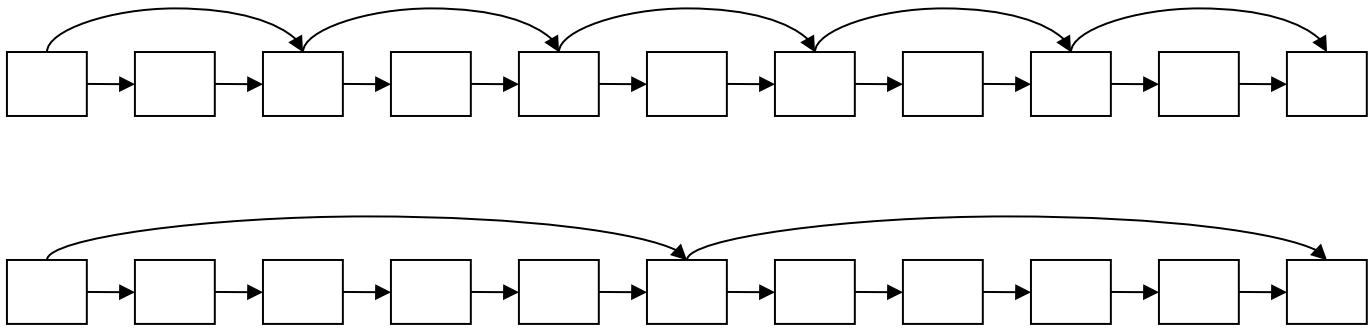
IntersectWithSkips(p_1, p_2)

```
1 answer ← <>
2 while p1 ≠ NIL and p2 ≠ NIL
3 do if docID(p1) = docID(p2)
4      then ADD(answer, docID(p1))
5           p1 ← next(p1)
6           p2 ← next(p2)
7      else if docID(p1) < docID(p2)
8           then if hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
9                then while hasSkip(p1) and (docID(skip(p1)) ≤ docID(p2))
10                    do p1 ← skip(p1)
11               else p1 ← next(p1)
12          else if hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
13               then while hasSkip(p2) and (docID(skip(p2)) ≤ docID(p1))
14                    do p2 ← skip(p2)
```

Where to place skips?

Tradeoff:

- More skips → shorter skip spans ⇒ more likely to skip. But lots of comparisons to skip pointers.
- Fewer skips → few pointer comparison, but then long skip spans ⇒ few successful skips.



Placing skips

- **Simple heuristic:** for postings of length P, use $\sqrt{P}$ evenly-spaced skip pointers.
- Easy if the index is relatively static; harder if P keeps changing because of updates.

Exercise – Skip Pointers

Exercise 2.6 [⋆]

We have a two-word query. For one term the postings list consists of the following 16 entries:

[4,6,10,12,14,16,18,20,22,32,47,81,120,122,157,180]

and for the other it is the one entry postings list:

[47].

Work out how many comparisons would be done to intersect the two postings lists with the following two strategies. Briefly justify your answers:

a. Using standard postings lists

b. Using postings lists stored with skip pointers, with a skip length of $\sqrt{P}$, as suggested in Section 2.3.

Exercise – Skip Pointers Answer

a) using standard postings lists

Compare the elements from 4 thru 47 in the first list. So total number of comparisons without skip pointers is 11.

b) using postings lists stored with skip pointers, with a skip length of sqrt (P)

The length of the list is 16 . So sqrt (16) = 4.

The skip pointers will be at the elements 4, 14,22 and 120

Compare 4 and 47, as 4 is less than 47 and also skip pointer, compare the skip pointer 14 with 47, then 22 with 47 and 120 with 47. As 120 is greater than 47, compare the element next to 22 , ie 32 , then 47.

So total 6 comparisons are done with skip pointers.

Exercise – Skip Pointers

Exercise 2.7 [⋆]

Consider a postings intersection between this postings list, with skip pointers:

and the following intermediate result postings list (which hence has no skip pointers):

3 5 89 95 97 99 100 101

Trace through the postings intersection algorithm in Figure 2.10 (page 37).

a. How often is a skip pointer followed (i.e., p_1 is advanced to $skip(p_1)$)?

b. How many postings comparisons will be made by this algorithm while intersecting the two lists?

c. How many postings comparisons would be made if the postings lists are intersected without the use of skip pointers?

Positional postings and Phrase queries

Phrase queries

- Want to be able to answer queries such as "***Stanford University***" – as a phrase
- The sentence *"I went to university at Stanford"* is not a match
- The sentence *"The inventor Stanford Ovshinsky never went to university"* is not a match
- For this, it no longer suffices to store only
  - <*term* : *docs*> entries

Handling Phrase queries

innovate achieve lead

- Biword Indexes
- Positional Indexes

Biword Indexes

- Index every consecutive pair of terms in the text as a phrase
- For example, the text “Friends, Romans, Countrymen” would generate the biwords
  - ***friends romans***
  - ***romans countrymen***
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

Biword indexes

- Longer phrases are processed by breaking them.
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:
  - ***stanford university*** *AND* ***university palo*** *AND* ***palo alto***

- **Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.**

Extended biwords

- Parse the indexed text and perform **part-of-speech-tagging** (POST).
- Bucket the terms into (say) Nouns (N) and articles/prepositions (X).
- Call any string of terms of the form NX*N an extended biword.
  - Each such extended biword is now made a term in the dictionary.
- Example: cost overruns on a power plant

  N N X X N
- Query processing: parse it into N's and X's
  - Segment query into extended biwords
  - "cost overruns" AND "overruns power" AND "power plant"
- Better results can be obtained by using more precise part-of-speech patterns that define which extended biwords should be indexed.

Issues for biword indexes

- False positives
- Index blowup due to bigger dictionary
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy.

- Blue house
- The sky was blue. House was at dark.

Positional indexes

- In the postings, store, for each ***term*** the position(s) in which tokens of it appear:

  - <***term***, number of docs containing ***term***;
  - *doc1*: frequency of the term; position1, position2 … ;
  - *doc2*: frequency of the term; position1, position2 … ;
  - etc.>

Positional indexes

to, 993427:
(1, 6: (7, 18, 33, 72, 86, 231);
2, 5: (1, 17, 74, 222, 255);
4, 5: (8, 16, 190, 429, 433);
5, 2: (363, 367);
7, 3: (13, 23, 191); . . .)
be, 178239:
(1, 2: (17, 25);
4, 5: (17, 191, 291, 430, 434);
5, 3: (14, 19, 101); . . .)

Positional indexes

- For phrase queries, we use a merge algorithm recursively at the document level
- This requires more than just equality
- Start with the least frequent term and then work, to further restrict the list of possible candidates
- Need to check that their positions of appearance in the document are compatible with the phrase query being evaluated
- This requires working out offsets between the words
- Positional Index can also be used for Proximity Searches.

Exercise - Positional Indexes

innovate achieve lead

Exercise 2.9 [⋆]

Shown below is a portion of a positional index in the format: term: doc1: ⟨position1, position2, … ⟩; doc2: ⟨position1, position2, … ⟩; etc.

angels: 2: ⟨36,174,252,651⟩; 4: ⟨12,22,102,432⟩; 7: ⟨17⟩;
fools: 2: ⟨1,17,74,222⟩; 4: ⟨8,78,108,458⟩; 7: ⟨3,13,23,193⟩;
fear: 2: ⟨87,704,722,901⟩; 4: ⟨13,43,113,433⟩; 7: ⟨18,328,528⟩;
in: 2: ⟨3,37,76,444,851⟩; 4: ⟨10,20,110,470,500⟩; 7: ⟨5,15,25,195⟩;
rush: 2: ⟨2,66,194,321,702⟩; 4: ⟨9,69,149,429,569⟩; 7: ⟨4,14,404⟩;
to: 2: ⟨47,86,234,999⟩; 4: ⟨14,24,774,944⟩; 7: ⟨199,319,599,709⟩;
tread: 2: ⟨57,94,333⟩; 4: ⟨15,35,155⟩; 7: ⟨20,320⟩;
where: 2: ⟨67,124,393,1001⟩; 4: ⟨11,41,101,421,431⟩; 7: ⟨16,36,736⟩;

Which document(s) if any match each of the following queries, where each expression within quotes is a phrase query?

a. "fools rush in"

b. "fools rush in" AND "angels fear to tread"

References

1. Alers-Valentín, H., Maldonado-Torres, J. D., & Vega-Riveros, J. F. (2025). Limitations of Tokenizers for Building a Neuro-Symbolic Lexicon. *SciTePress*.
2. Badawi, A. A., & Al-Haija, Q. A. (2013). IRS for Computer Character Sequences Filtration: a new software tool and algorithm to support the IRS at tokenization process. *International Journal of Advanced Computer Science and Applications(IJACSA), The Science and Information Organization (SAI), 4*(2).